A3: DESIGNING AND IMPLEMENTING JOBS TO NETWORK APPLICATION PROTOCOL USING TCP

Course: COMP 3670 – Computer Networks
Assignment 3 – Part 2
Group 11 Team Members

1. Charles Corro
2. Keertana Madhavan
3. Van Minh Ngai
4. Nitin Ramesh
5. Bilal Sohail

UNIVERSITY OF WINDSOR

Table of Contents

Introduction

In addition to our client-server architecture over TCP, we extended our network application to add the following features offered by the job creator. The following one-to-one services were offered to the job-seeker: Detect if a given IP address is online or not, detect the status of a given port at a given IP address. For the One-To-Many jobs, the following services were offered to the job-seeker: execute a TCP flood attack and execute a UDP flood attack against a given port on a given IP.

One to One Jobs

Online IP address:

In this feature the job creator would like the job seeker to find out if a given IP addresses is contacted the network or not. On the JobSeeker.java, the seeker would respond to the job creator request for the target's IP to find out it has contacted the network or not. In our code, when the creator received response "justip" then creator should enter the IP address and send it to the seeker. The seeker then will create the exec process to contact the machine to acquire a connection or respond and report the status of the IP address based on the getRunTime() method. After finishing the task, the seeker should send a "task done" message to the creator.

The Job Seeker code when it asks initiates the task to check for online machines

```
54        else if(response.equals("justip"))//option to check if given ip address is online
55        {
56          String ipA=is.readLine();
57          System.out.println("Checking if "+ipA+" is online...");
58          //Runtime.getRuntime().exec("ping " + ipA);
59          Runtime rt = Runtime.getRuntime();
60          Process proc = rt.exec("ping "+ipA);
61          BufferedReader stdInput = new BufferedReader(new InputStreamReader(proc.getInputStream()));
62          System.out.println("Here is the output of the command:\n");
63          String s = null;
64          String op3="";
65          while ((s = stdInput.readLine()) != null)
66          {
67            System.out.println(s);
68            op3=op3+s+"\n";
69          }
70          os.println(op3);
71          os.flush();
72          os.println("justip ~ DONE!");
73          os.flush();
74        }
```

Port Status:

In this feature, the job creator wants to detect the status of a given port of a given IP address. The status of the port will check for TCP and IDP port statuses. The creator will send the IP address and the port number to the seeker. The seeker then will initiate a Port scan to scan for open ports using netcat utility tool to read and write connections using TCP or UDP. After the scan, the contents of the scan and send the “task done” message to the creator.

The Job Seeker code when it asks initiates the task to check for a machine’s Port Status

```
75        else if(response.equals("portip"))//option to check if given IP's port's status
76        {
77          String ipA=is.readLine();
78          int portn=Integer.parseInt(is.readLine());
79          System.out.println("Checking status of port "+portn+" of "+ipA+" is online...");
80        //  telnet <ip_address> <port_number>
81          Runtime rt = Runtime.getRuntime();
82          Process proc = rt.exec("nc -zv "+ipA+" "+portn);
83          BufferedReader stdInput = new BufferedReader(new InputStreamReader(proc.getErrorStream()));
84          System.out.println("Here is the output of the command:\n");
85          String s = null;
86          String op3="";
87          while ((s = stdInput.readLine()) != null)
88          {
89            System.out.println(s);
90            op3=op3+s+"\n";
91          }
92          os.println(op3);
93          os.flush();
94          os.println("portip ~ DONE!");
95          os.flush();
96        }
```

One to Many Jobs

TCP Attack

In this feature, the job creator wants to ask more than one job seeker to execute a TCP flood attack against a. given IP and port number. The creator will send the IP address and the port number to the seeker. The seeker then will initiate a connection to the IP Address and send 1 message every second for a total of 60 seconds. After flooding the machine, the seeker will send a “task done” message to the creator.

The Job Seeker code when it asks initiates the task to initiate TCP Flood Attack

```
115         else if(response.equals("tcpattack"))//option to initiate TCP attack by making a socket with given IP address and port and opening a
116         {
117           String ipA=is.readLine();
118           int portn=Integer.parseInt(is.readLine());
119           try{
120             Socket sock=new Socket(ipA, portn); //establishes connection
121             PrintWriter inpst=new PrintWriter(sock.getOutputStream());//gets inputstream to flood with messages
122             System.out.println("TCP connection with victim established...");
123             for(int qk=0;qk<60;qk++)//sends 60 messages in a duration of 60 seconds
124             {
125               inpst.println("attacking...");
126               inpst.flush();
127               Thread.sleep(1000);
128             }
129
130           }
131           catch(Exception ex){
132             System.out.println("Unable to Establish connection...");
133           }
134           os.println("tcpattack ~ DONE!");
135           os.flush();
136         }
137                         else{
138                                 System.out.println("No message received");
139                         }
140           }
141         }
142       catch(IOException e)
143       {
144           e.printStackTrace();
145           System.out.println("Socket read Error");
146       }
147       finally
148       {
149           is.close();os.close();br.close();s1.close();
150           System.out.println("Connection Closed");
151       }
```

UDP Attack

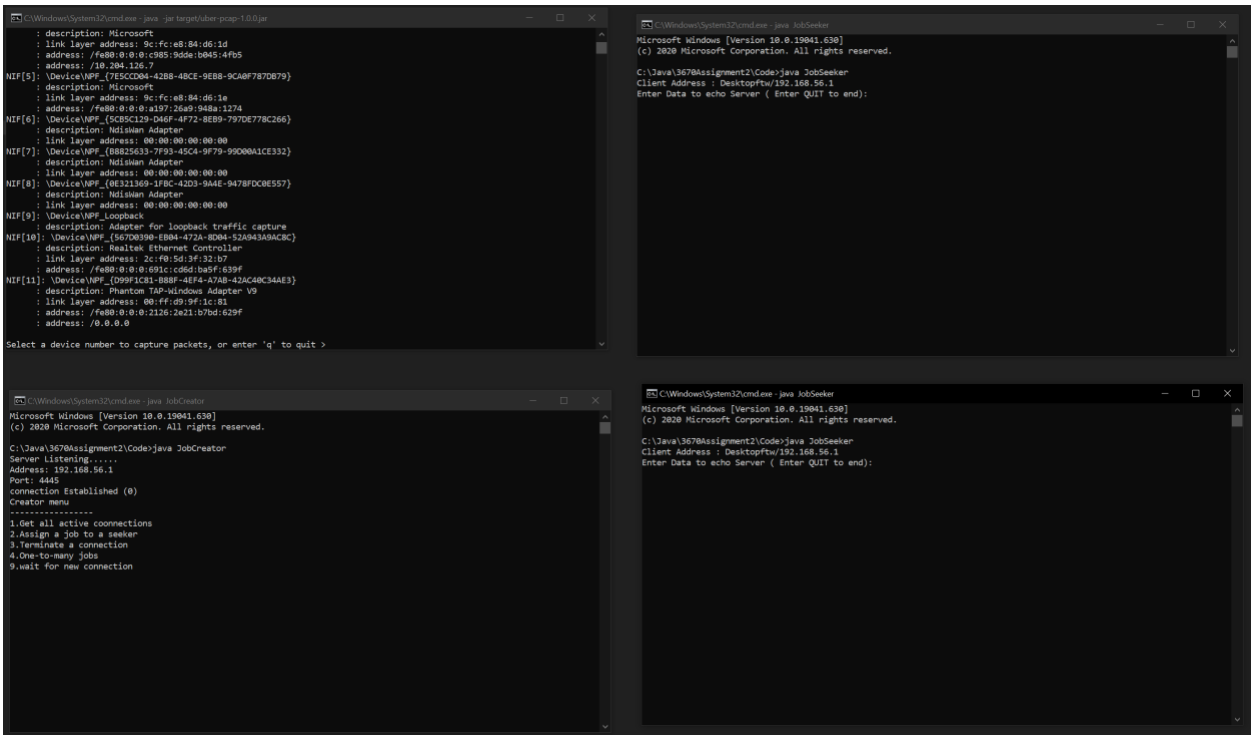
In this feature, the job creator wants to ask more than one job seeker to execute a UDP flood attack against a. given IP and port number. The creator will send the IP address and the port number to the seeker. The seeker then will initiate a connection to the IP Address and send 1 message every second for a total of 60 seconds. The DatagramSocket will be used to provide a connection-less point for receiving and sending packets. The DatagramPacket will be used to route each message to the machines. After flooding the machine, the seeker will send a “task done” message to the creator.

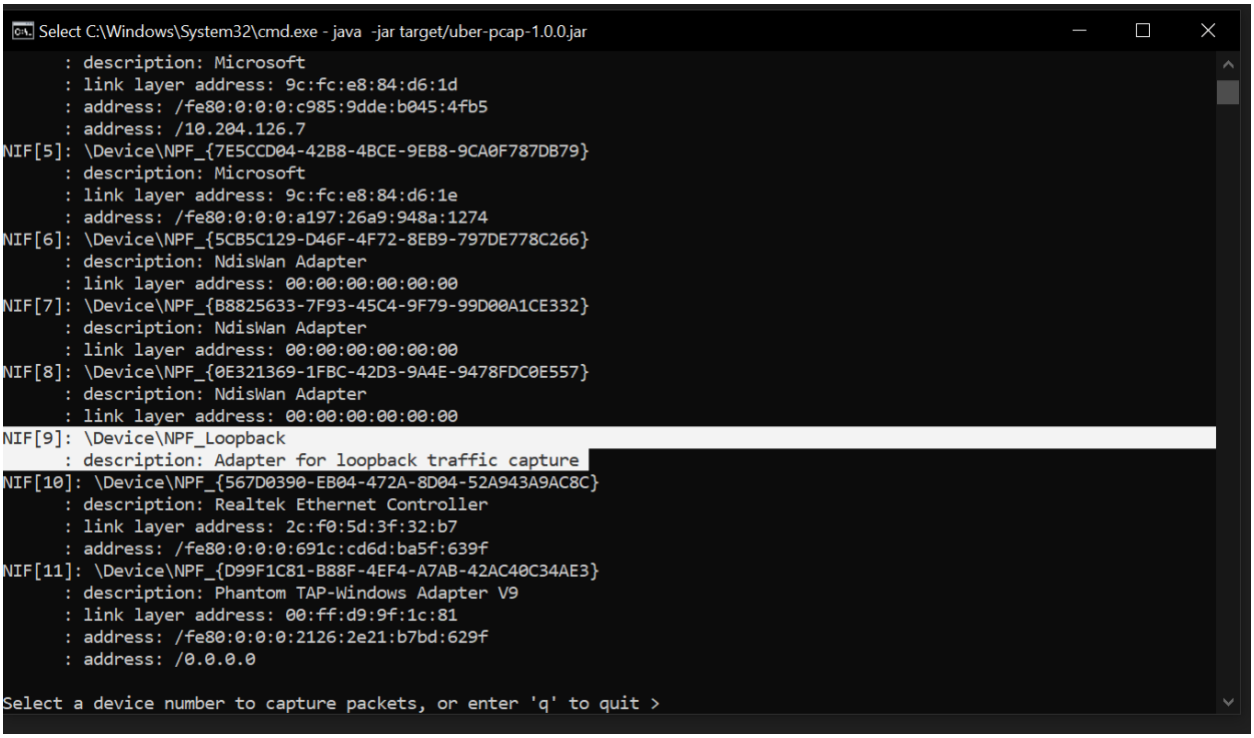
The Job Seeker code when it asks initiates the task to initiate a UDP Flood attack

```
else if(response.equals("udpattack"))//option to initiate UDP attack by pinging given IP address
{
  System.out.println("Initiating UDP attack...");
  String ipA=is.readLine();
  int portn=Integer.parseInt(is.readLine());
  InetAddress ip = InetAddress.getByName(ipA);
  DatagramSocket ds = new DatagramSocket();
  byte buf[] = "UDP Attack!".getBytes();
  DatagramPacket DpSend =  new DatagramPacket(buf, buf.length, ip, portn);
  for(int qk=0;qk<60;qk++)//sends 60 messages in a duration of 60 seconds
  {

     ds.send(DpSend);
    Thread.sleep(1000);
  }
  os.println("udpattack ~ DONE!");
  os.flush();
}
```

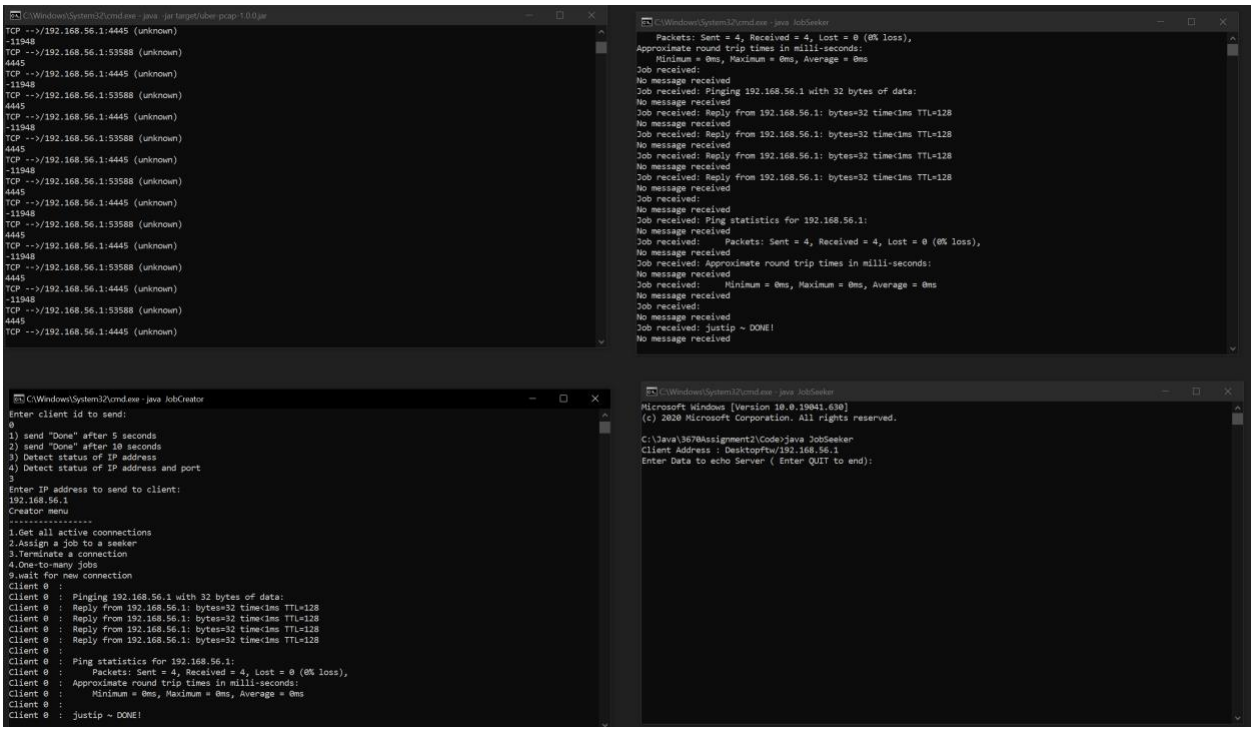
Testing and Screenshots



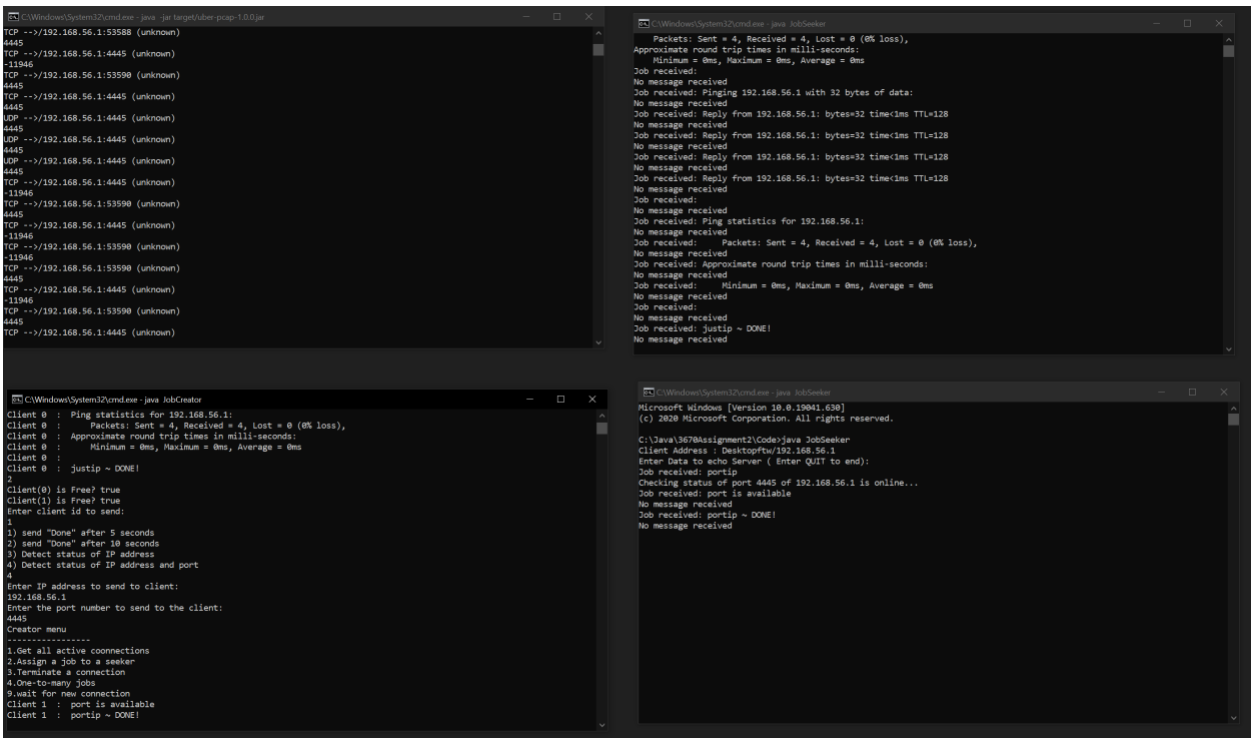


Here you need to download and set up the pcap directory from our repository that aids in creating the packets as shown in the screenshots above.

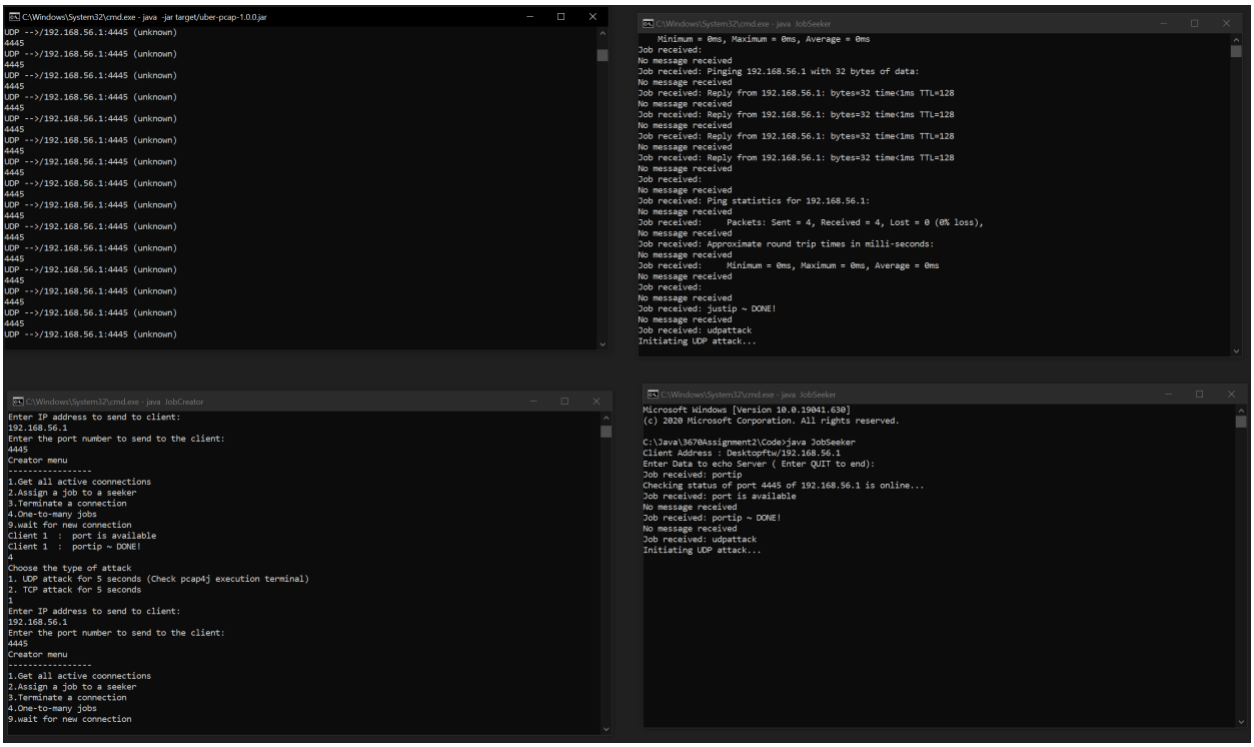
Result of scanning IP to check if online or not



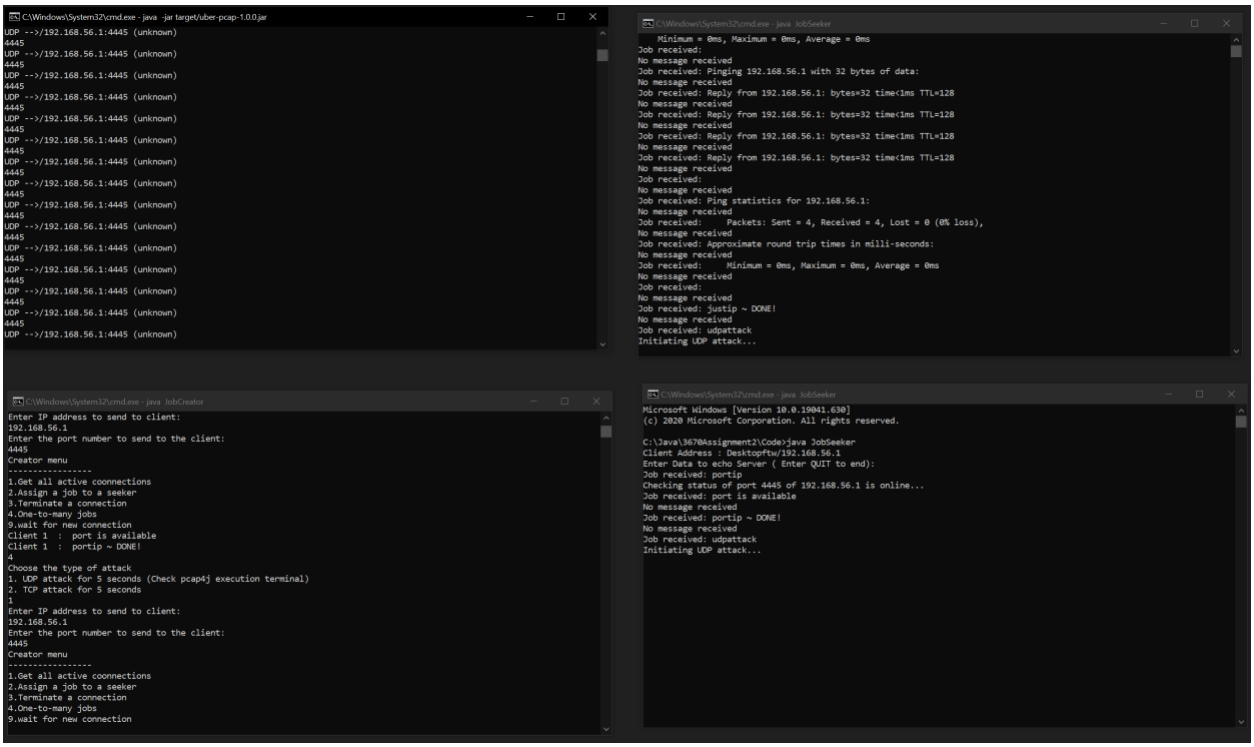
Result of scanning IP and port number to check for open ports (TCP and UDP)



Result of initiating an UDP Attack



Result of initiating an TCP attack


C:\Windows\System32\cmd.exe - java -jar target/uber-pcap-1.0.0.jar
UDP -->/192.168.56.1:4445 (unknown)
4445
C:\Windows\System32\cmd.exe - java JobSeeker
Minimum = 0ms, Maximum = 0ms, Average = 0ms
Job received:
No message received
Job received: Pinging 192.168.56.1 with 32 bytes of data:
Job received: Reply from 192.168.56.1: bytes=32 time<1ms TTL=128
Job received: Ping statistics for 192.168.56.1:
Job received: Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Job received: Approximate round trip times in milli-seconds:
Job received: Minimum = 0ms, Maximum = 0ms, Average = 0ms
Job received: justip ~ DONE!
Job received: udpattack
Initiating UDP attack...
C:\Windows\System32\cmd.exe - java JobCreator
Enter IP address to send to client:
192.168.56.1
Enter the port number to send to the client:
4445
Creator menu
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
4.One-to-many jobs
9.wait for new connection
Client 1 : port is available
Client 1 : portip ~ DONE!
Choose the type of attack
1. UDP attack for 5 seconds (Check pcap4j execution terminal)
2. TCP attack for 5 seconds
C:\Windows\System32\cmd.exe - java JobSeeker
Microsoft Windows [Version 10.0.19041.630]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server (Enter QUIT to end):
Job received: portip
Checking status of port 4445 of 192.168.56.1 is online...
Job received: port is available
Job received: portip ~ DONE!
Job received: udpattack
Initiating UDP attack...

Code

Packet Capturing PCAP Directory

```
// App.java
package com.github.username;

import java.io.IOException;
import org.pcap4j.core.*;
import org.pcap4j.util.*;
import org.pcap4j.packet.*;
import java.util.*;
import org.pcap4j.core.PcapNetworkInterface.PromiscuousMode;
import org.pcap4j.core.BpfProgram.BpfCompileMode;
public class App
{
 int packet_count = 0;
  private static final String READ_TIMEOUT_KEY =
PacketListener.class.getName() + ".readTimeout";
  private static final int READ_TIMEOUT =
Integer.getInteger(READ_TIMEOUT_KEY, 10); // [ms]
  private static final String SNAPLEN_KEY = PacketListener.class.getName() +
".snaplen";
  private static final int SNAPLEN = Integer.getInteger(SNAPLEN_KEY, 65536);
// [bytes]

  private static final String TIMESTAMP_PRECISION_NANO_KEY =
PacketListener.class.getName() + ".timestampPrecision.nano";
  private static final boolean TIMESTAMP_PRECISION_NANO =
Boolean.getBoolean(TIMESTAMP_PRECISION_NANO_KEY);

public int sniff() throws PcapNativeException, NotOpenException
{

    PcapNetworkInterface nif = null;
    try {
      nif = new NifSelector().selectNetworkInterface();
    } catch (IOException e) {
      e.printStackTrace();
    }

    if (nif == null) {
      return -1;
    }

    System.out.println(nif.getName() + "(" + nif.getDescription() + ")");

    PcapHandle handle = nif.openLive(65536, PromiscuousMode.PROMISCUOUS, 10);
       handle.setFilter("tcp port 4445 or udp port 4445",
BpfCompileMode.OPTIMIZE);
    while (true)
    {
      Packet packet = handle.getNextPacket();
      if (packet != null)
      {
        if (packet.contains(IpV4Packet.class) &&
(packet.contains(TcpPacket.class) || packet.contains(UdpPacket.class)))
```

```
        {
          IpV4Packet pkt = packet.get(IpV4Packet.class);
          if (packet.contains (TcpPacket.class))
           {
             TcpPacket tcpPkt = packet.get(TcpPacket.class);
        System.out.println(tcpPkt.getHeader().getDstPort().value());

                 System.out.println("TCP -->"+pkt.getHeader().getSrcAddr() +
":" + tcpPkt.getHeader().getDstPort());
            }

          else
          {
             UdpPacket udpPkt = packet.get(UdpPacket.class);
        System.out.println(udpPkt.getHeader().getDstPort().value());

             System.out.println("UDP -->"+pkt.getHeader().getSrcAddr() + ":"
+ udpPkt.getHeader().getDstPort());
          }


         }
        else
        {
                System.out.println("Unknown packet ---> "+
packet.getPayload());
         }
       packet_count++;
      }
    }
  }
    public static void main(String[] args) throws PcapNativeException,
NotOpenException
    {
        App o1=new App();
         o1.sniff();
}
}
```